执行 Babel 生成的代码

即便你已经用 Babel 编译了你的代码，但这还不算完。

babel-polyfill

Babel 默认只转换新的 JavaScript 句法（syntax），而不转换新的 API，比如 Iterator、Generator、Set、Maps、Proxy、Reflect、Symbol、Promise 等全局对象，以及一些定义在全局对象上的方法（比如 Object.assign）都不会转码。Babel 默认不转码的 API 非常多，详细清单可以查看 definitions.js 文件。

举例来说，ES6 在 Array 对象上新增了 Array.from 方法。Babel 就不会转码这个方法。如果想让这个方法运行，必须使用 babel-polyfill，为当前环境提供一个垫片。

比方说，我们需要编译以下代码：

```
function addAll() {
  return Array.from(arguments).reduce((a, b) => a + b);
}
```

最终会变成这样：

```
function addAll() {
  return Array.from(arguments).reduce(function(a, b) {
    return a + b;
  });
}
```

然而，它依然无法随处可用因为不是所有的 JavaScript 环境都支持 `Array.from`。

```
Uncaught TypeError: Array.from is not a function
```

为了解决这个问题，我们使用一种叫做 Polyfill（代码填充，也可译作兼容性补丁） 的技术。 简单地说，polyfill 即是在当前运行环境中用来复制（意指模拟性的复制，而不是拷贝）尚不存在的原生 api 的代码。 能让你提前使用还不可用的 APIs，`Array.from` 就是一个例子。
Babel 用了优秀的 core-js 用作 polyfill，并且还有定制化的 regenerator 来让 generators（生成器）和 async functions（异步函数）正常工作。

要使用 Babel polyfill，首先用 npm 安装它：

```
$ npm install --save babel-polyfill
```

然后只需要在文件顶部导入 polyfill 就可以了：

```
import "babel-polyfill";
```

babel-runtime

为了实现 ECMAScript 规范的细节，Babel 会使用“助手”方法来保持生成代码的整洁。

由于这些助手方法可能会特别长并且会被添加到每一个文件的顶部，因此你可以把它们统一移动到一个单一的“运行时（runtime）”中去。

通过安装 `babel-plugin-transform-runtime` 和 `babel-runtime` 来开始。

```
$ npm install --save-dev babel-plugin-transform-runtime
$ npm install --save babel-runtime
```

然后更新 `.babelrc`：

```
  {
    "plugins": [
+     "transform-runtime",
      "transform-es2015-classes"
    ]
  }
```

现在，Babel 会把这样的代码：

```
class Foo {
  method() {}
}
```

编译成：

```
import _classCallCheck from "babel-runtime/helpers/classCallCheck";
import _createClass from "babel-runtime/helpers/createClass";

let Foo = function () {
  function Foo() {
    _classCallCheck(this, Foo);
  }

  _createClass(Foo, [{
    key: "method",
    value: function method() {}
  }]);

  return Foo;
}();
```

这样就不需要把 `_classCallCheck` 和 `_createClass` 这两个助手方法放进每一个需要的文件里去了。